# **Python For Data Science** Cheat Sheet

# Keras

Learn Python for data science Interactively at www.DataCamp.com



#### Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

## A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2, size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch size=32)
>>> predictions = model.predict(data)
```

#### Data

#### Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the train test split module of sklearn.cross validation.

#### Keras Data Sets

```
>>> from keras.datasets import boston housing,
                                    cifar10.
                                    imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x train2, y train2), (x test2, y test2) = boston housing.load data()
>>> (x_train3,y_train3), (x_test3,y_test3) = cifar10.load_data()
>>> (x train4, y train4), (x test4, y test4) = imdb.load data(num words=20000)
>>> num classes = 10
```

#### Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

# Model Architecture

## Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

#### Multilayer Perceptron (MLP)

#### **Binary Classification**

```
>>> from keras.lavers import Dense
>>> model.add(Dense(12,
                     input dim=8,
                     kernel initializer='uniform'.
                     activation='relu'))
>>> model.add(Dense(8,kernel initializer='uniform',activation='relu'))
>>> model.add(Dense(1, kernel initializer='uniform', activation='sigmoid'))
Multi-Class Classification
```

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

>>> model.add(Dense(64,activation='relu',input dim=train data.shape[1])) >>> model.add(Dense(1))

>>> from keras.layers import Activation, Conv2D, MaxPooling2D, Flatten

### Convolutional Neural Network (CNN)

```
>>> model2.add(Conv2D(32,(3,3),padding='same',input shape=x train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool size=(2,2)))
>>> mode12.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> mode12.add(Dropout(0.5))
>>> model2.add(Dense(num classes))
```

## >>> model2.add(Activation('softmax')) Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

# Preprocessing

## Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x train4 = sequence.pad sequences(x train4, maxlen=80)
>>> x test4 = sequence.pad sequences(x test4, maxlen=80)
```

#### One-Hot Encoding

```
>>> from keras.utils import to categorical
>>> Y train = to categorical(y train, num classes)
>>> Y test = to categorical(y test, num classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y test3 = to categorical(y test3, num classes)
```

#### **Train and Test Sets**

```
>>> from sklearn.model selection import train test split
>>> X train5, X test5, y train5, y test5 = train test split(X,
                                                             test_size=0.33,
random state=42)
```

Also see NumPy & Scikit-Learn

# Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x train2)
>>> standardized X = scaler.transform(x train2)
>>> standardized X test = scaler.transform(x test2)
```

#### **Inspect Model**

```
Model output shape
>>> model.output shape
>>> model.summary()
                                     Model summary representation
>>> model.get config()
                                     Model configuration
>>> model.get weights()
                                     List all weight tensors in the model
```

# **Compile Model**

```
MLP: Binary Classification
>>> model.compile(optimizer='adam',
                   loss='binary crossentropy',
                   metrics=['accuracy'])
MLP: Multi-Class Classification
>>> model.compile(optimizer='rmsprop',
                   loss='categorical crossentropy',
                   metrics=['accuracy'])
MLP: Regression
>>> model.compile(optimizer='rmsprop',
                   loss='mse'.
                   metrics=['mae'])
```

#### Recurrent Neural Network

```
>>> model3.compile(loss='binary crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

## **Model Training**

```
>>> model3.fit(x train4,
              v train4,
              batch size=32,
             epochs=15,
              verbose=1,
             validation data=(x test4, y test4))
```

## **Evaluate Your Model's Performance**

```
>>> score = model3.evaluate(x test,
                                 y_test,
batch size=32)
```

#### Prediction

```
>>> model3.predict(x test4, batch size=32)
>>> model3.predict classes(x test4,batch size=32)
```

# Save/Reload Models

```
>>> from keras.models import load model
>>> model3.save('model file.h5')
>>> my model = load model('my model.h5')
```

# **Model Fine-tuning**

## Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

## Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early stopping monitor = EarlyStopping(patience=2)
>>> model3.fit(x train4,
             y train4,
             batch size=32,
             epochs=15,
             validation data=(x test4, y test4),
             callbacks=[early stopping monitor])
```

#### **DataCamp Learn Python for Data Science Interactively**

